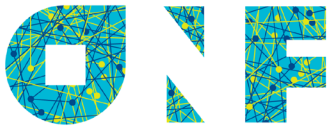

OPEN NETWORKING
FOUNDATION

Wireless Transport SDN Proof of Concept 2 Detailed Report

Version 1.0
June 6, 2016

ONF Document Type: White Paper
ONF Document Name: Wireless Transport SDN PoC White Paper

Disclaimer

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

Table of Contents

Executive Summary **5**

1 Introduction **5**

2 SDN Network Architecture and Configuration **6**

2.1 Overview 6
2.2 PoC Test Network Setup 7

3 Use Cases and Applications **8**

4 Test Results **9**

5 Conclusions **10**

6 Contributions **11**

6.1 Functional Model 11
6.2 UML Information Model 12
6.3 YANG Data Model 14
6.4 Evaluation of Open Source Netconf Servers 15
6.5 Basis Mediator 16
6.6 Default Value Mediator 18
6.7 Element Auto Detection 19
6.8 UI for Showing and Configuring Parameters 21
6.9 Application for Showing Aberrances 23
6.10 GUI for Showing the Configured Network 25
6.11 GUI for Showing the Effective Network 28
6.12 Event Handling 29
6.13 Testing Framework 31

7 References **33**

List of Figures

Figure 1: Overview of the SDN Architecture used in the 2nd Wireless Transport PoC 6

Figure 2: SDN Test Network Configuration using Virtual Machines 8

Figure 3: Example Functional Model for plain Microwave Link 12

Figure 4: Main Elements of the Microwave Model 13

Figure 5: Example of a Pac 13

Figure 6: Basic Mediator 16

Figure 7: netconfd Server (from open Yuma user manual) 17
Figure 8: Tools of OpenYuma (from open Yuma user manual) 18
Figure 9: Mediator Discovery Sequence of Operations 21
Figure 10: Configuration GUI 23
Figure 11: Path from Planning to Actual Object 24
Figure 12: Aberrances User Interface 25
Figure 13: Configured Network GUI 27
Figure 14: Explanation of Configured Network GUI 27
Figure 15: Effective Network GUI 28
Figure 16: Event Handling in OpenDaylight 30
Figure 17: Testing Framework Architecture 32

Executive Summary

This white paper provides an overview of a significant technical Proof of Concept (PoC) project conducted from 25th to 28th of April 2016 by the Wireless Transport Project of the Open Networking Foundation (ONF) in Munich, Germany.

This PoC focused on demonstrating the capabilities and benefits of utilizing a common Information Model for multi-vendor control of wireless network elements through open management interfaces.

The PoC included wide participation from the wireless transport industry including operator representatives, microwave equipment vendors, integrators and applications providers.

Five uses cases are included in this PoC to demonstrate wireless transport SDN applications illustrating topology planning and discovery, with dynamic view in real time, configuration, discrepancy monitoring and detection, and event handing.

The following 5 use cases are implemented and are the subject of this PoC for the purpose of demonstrating the aforementioned applications and presented in this whitepaper:

- Detection and configuration of new microwave devices
- Detection of aberrances
- Detection and Visualization of the configured microwave network
- Detection and Visualization of the currently effective network
- Receiving, displaying and storing of alarm information

A standard OpenDaylight (ODL) version was used as the SDN controller. Mediators were used for translating the information model to vendor specific configurations.

All vendors implemented the model and completed all the test cases successfully demonstrating the viability of the concept for using a common information model for configuring and management of wireless network elements using open management interfaces.

1 Introduction

The core element of the 2nd wireless transport PoC is to prove the effectiveness of an SDN wireless transport modeling in a multivendor radio transport network.

The 2nd wireless transport PoC is an evolution of the 1st one, emphasizing the relevance of a common information model being developed in ONF by the Wireless Transport Project. The intended use cases leverage on the information model to offer a common and stable framework for multi-vendor control of wireless network elements through open management interfaces like Netconf.

This incremental step aims to demonstrate advantages in the operation of wireless transport networks incorporating such open capabilities, and show the facility of integration and cooperation with other parts of the network.

The 2nd wireless transport PoC took place from 25th to 28th of April 2016 and was hosted by Telefónica Germany in their laboratory in Munich Germany.

The PoC was supported by wide representatives from the wireless transport eco system including operators, manufacturers, integrators and applications providers. In particular, the following companies supported the PoC:

- The following Vendors participated in the PoC with their microwave equipment: Ceragon, Ericsson, Huawei, NEC and SIAE
- The following Integrators and Application Providers provided buildings block and applications:
  HCL, highstreet technologies, Tech Mahindra and Wipro
- Content and organizational support for the PoC was provided by the following Operators: AT&T, Deutsche Telekom and Telefónica (Telefonica Germany [1] and Telefónica I+D/Telefónica Global CTO[2])

Moreover, Viavi provided measurement devices and configuration support.

2 SDN Network Architecture and Configuration

2.1 Overview

The SDN architecture and configuration of the test setup in the 2nd Wireless Transport PoC is illustrated in Figure 1 below.

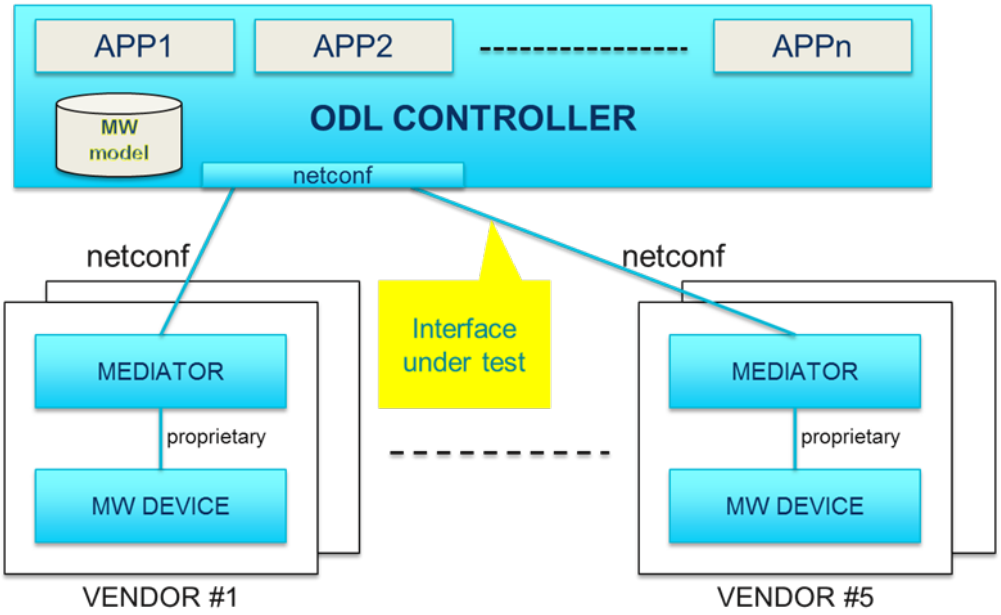

Figure 1: Overview of the SDN Architecture used in the 2nd Wireless Transport PoC

[1] As host and provider of lab facilities and logistics
[2] As part of the 5G-Crosshaul project (grant no. 671598 of EU H2020 program; *www.5g-crosshaul.eu*)

The architecture identifies the main objects which are target of implementation and subsequent verification: The ODL controller, the mediators, the Netconf interface and the northbound applications.

The network deployment includes a single SDN controller, an application layer which implements specific functions that are intended to operate over the network via northbound interfaces and a wireless network layer. The wireless network layer is composed of pairs of devices and mediators and interoperates with the controller via Netconf. The mediators are devices specific and proprietary to the vendors.

The developed functional information model is incorporated into OpenDaylight via Netconf/YANG plugins able to populate the MD-SAL data store.

- The OpenDaylight has been object of implementation to allow up streamed message exchange
- Further packages have been implemented atop the ODL DLUX GUI bundle
- Microwave applications can be installed and run on top of the extended OpenDaylight SDN controller with the microwave plugins

A testing framework has been also developed for end-to-end solution verification in a continuous integration perspective.

Netconf has been chosen over OpenFlow as a protocol for configuration and management of the microwave devices for several reasons. It is by design a general-purpose management protocol, while OpenFlow is primarily intended for operating the traffic forwarding plane of the device (e.g. traffic flows). Also, YANG can be used to implement the data model for Netconf, coming to a representation of a clearer and more readable information model. Instead of implementing Netconf protocol handlers in the devices, some external mediators (adapters) were used for translating the Netconf/YANG microwave information model to/from the existing proprietary management protocols of each vendor's devices. This approach allowed:

- being more flexible
- saving time in the development and the debugging phase
- sharing code between PoC participants

In addition to the mediators required to connect the physical NEs, an NE simulator ("Default Values Mediator") has been developed. It behaves like a generic NE, which allows re-demonstrating the use cases and applications from the PoC without the actual need and installation of physical microwave equipment. This supports Telefonica's plans on maintaining a server running a demonstration and testing environment for future application developments.

2.2 PoC Test Network Setup

The SDN network of the PoC is implemented using Virtual Machines (VM) for the core SDN controller – including some management applications – external applications, and for the mediators used by each vendor for integrating into their hardware. The network configuration with the VM is shown in Figure 2.

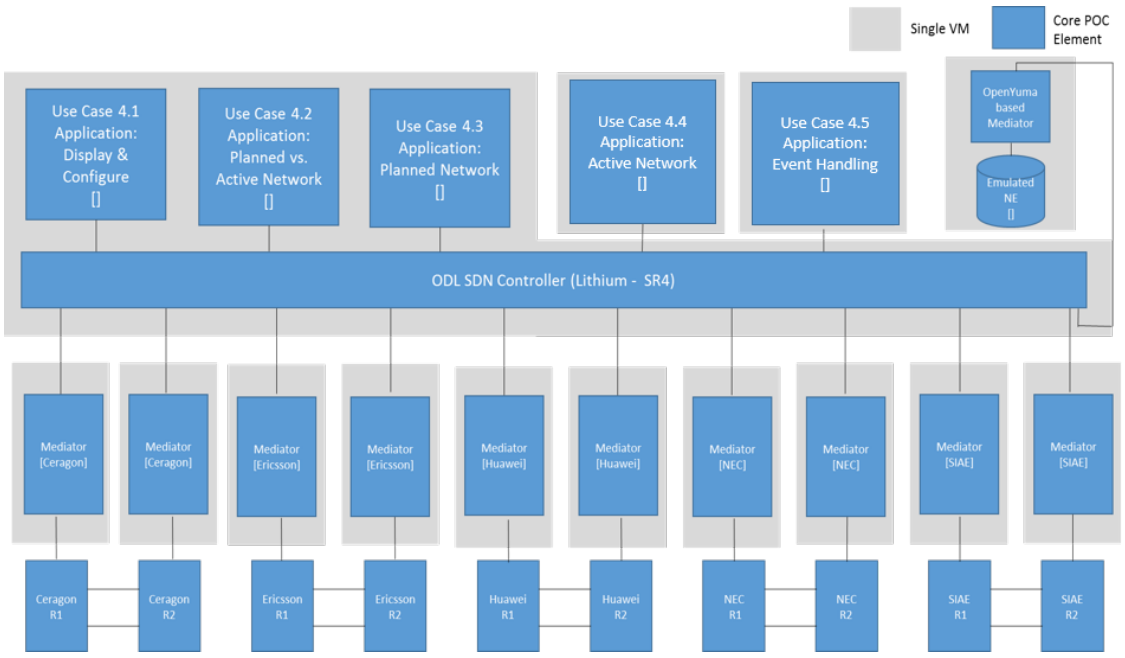

Figure 2: SDN Test Network Configuration using Virtual Machines

3 Use Cases and Applications

The following use cases are implemented for the purpose of demonstrating dynamic network view, configuration, discrepancy monitoring and detection, and event handing:

- Detection and configuration of new microwave devices
- Detection of aberrances
- Detection and Visualization of the configured microwave network
- Detection and Visualization of the currently effective network
- Receiving, displaying and storing of alarm information

Brief description of each use case is provided in the subsections below.

Detection and configuration of new microwave devices

This use case is used to demonstrate the capability of the SDN controller in detecting and configuring new network elements (NE) when introduced and deployed into the network. The Controller is used to collect information from the new NE and provide up-to-date information about the network. It achieves this by automatically identifying new elements added to the network and displaying current element configurations. The use case also demonstrates the capability of manually configuring a NE.

Detection of aberrances

This use case illustrates the Controller and its corresponding Application the capability of comparing the actual network configuration with external reference data such as the planned configuring for an NE. The controller can then identify discrepancies between the planned and

current configurations and present the network operator with corrections, which can then be confirmed and executed into the network.

Detection and Visualization of the configured microwave network

This use case relates to demonstrating the capability of the Controller and its Application in detecting the installed and deployed NEs and display the desired underlying wireless transport network topology then calculate and configure the capacity information for each NE based on the provided configuration information. The controller can then display the configured topology and related capacity information of the network at each microwave link.

Detection and Visualization of the currently effective network

This use case demonstrates the capability of the controller to detect the effective network topology, the installed and active NEs, and their corresponding effective capacity. It also demonstrates application for which the controller can detect and display deviations from the planned and configured network to the actually current network status.

Receiving, displaying and storing of alarm information

This last use case is introduced in the PoC to demonstrate the capabilities of the Controller in handling alarms and events from the network. This is demonstrated by storing and displaying events information when they occur. The information used in the PoC for each event includes NE IDs, type of problem, severity identifier and timestamp.

4 Test Results

Five wireless transport vendors and four integrators/application providers participated successfully in the test. The tests were done with an OpenDaylight (Lithium SR4) SDN controller. Additional functions required for the PoC use cases were provided by various SDN applications which had been implemented for the PoC and integrated into the SDN controller:

- Topology Application
- Comparison Application
- Configuration Application
- Event Handling Application
- NE Discovery application

The same south bound interface (Netconf protocol and YANG data models) was used between the OpenDaylight controller and the mediators from different vendors. The interfaces between mediators and network elements were proprietary interfaces which were not included for testing in this PoC.

The five use cases explained in section 3 were successfully tested by performing the following steps for the network elements of each vendor.

After having been started, the mediators announced their network elements to the Controller through the NE Discovery Application. The network elements became visible at the Controller GUI. The Topology Application displayed the configured capacity of all radio links in a graphical representation of the PoC network. The Comparison Application showed the planned

and actual values of all microwave attributes and highlighted the discrepancies. For the purpose of the PoC discrepancies were provoked by giving “wrong” planned values.

The configuration of the network elements was then modified by using the Configuration Application: either the transmitter of one link was switched off or the transmission power was reduced. As a consequence the affected link provided less or no capacity and the overall capacity of the network was reduced. This effect could be seen both in the Topology Application and at the GUI of the traffic analyzer. The original capacity was restored as soon as the respective configuration was reverted to the initial values.

The configuration changes also triggered the network elements to report events which were forwarded as Netconf notifications to the Controller. The Event Handling Application was used to display these notifications.

5 Conclusions

Operators

This Second PoC has been proved to be a significant step towards the generalization of a common Information Model for SDN-enabled Wireless Transport environments, simplifying the operations and control of these network elements, and facilitating the integration of distinct multi-vendor solutions under a common and single control framework.

It has been demonstrated a unified control and management of five wireless transport vendor products (Ceragon, Ericsson, Huawei, NEC, SIAE) implementing the same Information Model (being defined in the context of ONF by the Wireless Transport project, with the participation of those major manufacturers), in an open environment with an OpenDaylight based controller.

Vendors

The PoC has demonstrated how it’s possible to cooperate to provide a multi-vendor microwave network managed by a single SDN controller. This offers the possibility to operators for developing their own applications on top of an open and modern REST northbound interface, independently from the vendors providing the microwave infrastructure.

In addition, the concept of ‘mediator’ has allowed the possibility to integrate into this new network architecture, SDN based, also products that natively are not providing a Netconf interface. Indeed ‘mediator’ exposes a ‘Netconf’ interface towards the SDN Controller and provides mapping of the standardized Netconf command into the vendor-specific interface commands (for example SNMP).

Integrators and Application Providers

The demonstrated application has shown that all vendors implemented a Netconf common southbound management interface based on a YANG model which includes events handling.

Such common information and data modeling between network elements and SDN controller made the integration of NEs into SDN controllers very efficient. However, while syntax for models (based on YANG) can be considered a well addressed topic, semantics needs further attention. An ONF TR document should take care of the definition and the harmonization of the semantics.

Since the diverse PoC components needed an appropriate verification of the quality of the implementation, a simulator and/or an integrated testing environment have been reckoned as mandatory; it has been therefore developed. In fact, several portions have been object of custom implementations during the PoC development: the OpenDaylight itself, as well as the frameworks/environments for Netconf management used in the mediators (Netopeer, OpenYuma, ConfD) might be mentioned. The simulator environment might be leveraged as a PoC showcase, beyond the development phase.

Tight communication between vendor development, system integrators, application developers and service providers has been required to focus on the required functionality and use cases.

The choice of OpenDaylight as the SDN Controller has been a pondered decision, based on its inherent features in conjunction with the stability and maturity of its current implementation. OpenDaylight is in fact a well-documented project: it facilitates the addition of internal features and bundles or plugin of external applications. Standard technologies and frameworks like Karaf, Maven, Java, angular.js (among the others) allow software developers to adopt and improve existing code fragments.

Development of a PoC system provided us with many benefits including, among others, the:

- Very clear understanding of the information model
- Understanding of the capabilities and limitations of YANG and its support in OpenDaylight
- Ability to assess design decisions for later PoCs and products

6 Contributions

6.1 Functional Model

Context

A functional model is used to describe a network element by specifying a library of basic building blocks. The specification method is based on the functional decomposition of the equipment into atomic and compound functions. The functional model is also the basis of the information model.

Implementation

Functional models of transport network elements are usually defined as Recommendations in ITU-T, e.g. OTN functional model is defined in ITU-T Rec G.798. However, for microwave link, there's no such recommendation as the network elements of one microwave hop are always from the same vendor without interoperability.

In order to have a common understanding of microwave network elements, a functional model of microwave was discussed in Wireless Transport Project. This functional model specifies the functional blocks for a basic microwave network element, and also some complex functions such as Frequency Diversity (FD), Space Diversity (SD), protection, and Link Group survivability. The figure below shows the functional model for a basic microwave network element, which describes how the client signal is transmitting through the microwave link.

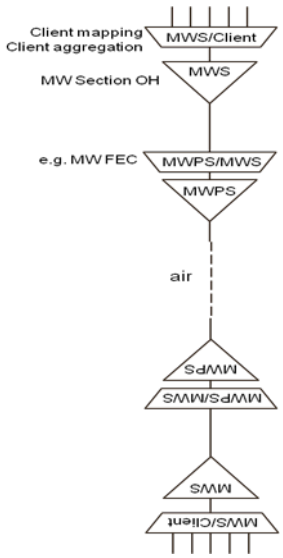

Figure 3: Example Functional Model for plain Microwave Link

Resources

The microwave functional model is available for download at the ONF ARO platform (https://login.opennetworking.org/bin/c5i?mid=4&rid=5&gid=0&k1=1815&tid=1462764447).

6.2 UML Information Model

Context

Powerful open interfaces are the basic element of a successful open ecosystem. Using a standardized interface protocol (e.g. Netconf, SNMP) is a prerequisite, also the information, which is exchanged, has to be unambiguously described and published.

An information model defines the information itself, but it also groups information and describes relationships between segments. It is a generic description and can be used to define interfaces as well as data structures inside programs.

A powerful open information model is capable of addressing all functionalities available at a type of device independently from the model or manufacturer.

Implementation

The microwave information model, which has been defined within the Wireless Transport Project, describes the analog characteristics of the air interface (e.g. frequency, channel bandwidth …), including XPIC, MIMO, hot stand-by and diversity configurations.

It segments the physical resources for transporting Ethernet and TDM traffic, and allows distributing flows across several air interfaces (e.g. Layer 1 Link Bonding).

Its ~300 attributes comprise a very complete set of configuration parameters plus performance indicators and a basic set of problem notifications. PDH, SDH, Synchronization, Data Connection Network (DCN) and user management are not yet covered.

Main elements:

- AirInterface – Represents the lowest layer in the microwave model, all the physical characteristics of the single microwave air interface are attached to this layer
- Structure – Segments the physical resource provided by the AirInterface into logical pieces that can be booked by Containers
- Container – Offers transport service to higher protocol layers, e.g. fixed size containers for TDM or containers of flexible size for Ethernet

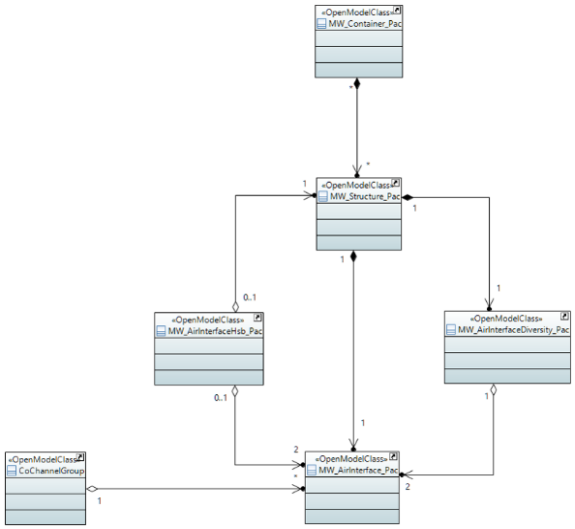

Figure 4: Main Elements of the Microwave Model

The microwave model is consolidated into the ONF core information model by associating its classes to the LayerProtocol class. The Pacs are structured into capabilities, configuration, status, problem and performance information.

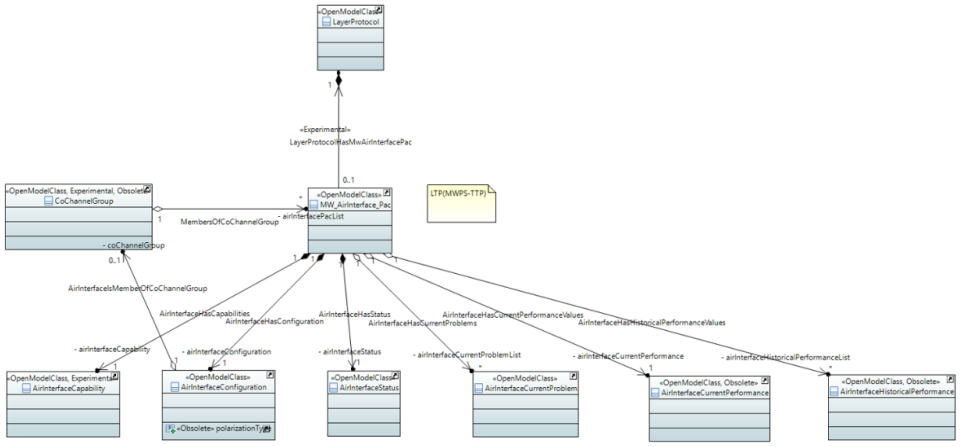

Figure 5: Example of a Pac

During the PoC a subset of ~60 priority one attributes has been applied.

Resources

Papyrus, which is a plug-in to the integrated development environment Eclipse, has been used for describing the information model. Eclipse (https://www.eclipse.org/) as well as Papyrus (http://www.eclipse.org/papyrus/) are freely available. ONF is providing a good Papyrus Guideline to get started

(https://www.opennetworking.org/search-result?searchword=papyrus%20guidelines&ordering=newest&searchphrase=all&limit=50).

The information model has been described in Unified Modeling Language (UML). It is available as project export from Papyrus and MS Word document at http://www.openmicrowave.com/?p=42&preview=true. UML is very generic and can easily be translated into many other languages as for example Java or YANG. A derivative YANG data model can simply be loaded into a Netconf interface.

Microwave is just a very little segment of the ONF's standardization work. All the contributed technology specific information models have to consolidate into the ONF core information model. ONF is providing the Open Model Profile and a very helpful ONF UML Guideline at

https://www.opennetworking.org/search-result?searchword=UML%20guidelines&ordering=newest&searchphrase=all&limit=50.

6.3 YANG Data Model

Context

YANG is a data modelling language for the Network Configuration Protocol (Netconf) and the Restconf applications configuration protocol.

Netconf is mainly used as management protocol between SDN controllers and network elements (southbound). Restconf is used as configuration protocol between SDN applications and SDN controller (northbound). Due to the architecture of OpenDaylight's MD-SAL (model driven service abstraction layer), YANG became also a modelling language for the data store.

YANG modules are also used for validation.

The YANG model, which has been used in the PoC, is derived from the UML model of the ONF core and its microwave specific extensions.

Implementation

The ONF has a project called Eagle, which offers a JavaScript (node.js) tool for converting from UML to YANG. UmlYangTools are currently under development, however they already provide enough functionality to convert a reduced ONF core model 1.1 including microwave extensions to YANG.

The reduction of the core model was necessary to focus only on the use cases for the PoC. The core model requires the object classes "NetworkElement", "LogicalTerminationPoint" and "LayerProtocol" to define the configuration protocol between network elements and SDN controller. In future, the object classes forming the "Forwarding Construct" will be needed to support protection use cases.

Object classes and attributes of the microwave information model, which are not of priority one, were tagged as "obsolete" in the UML and filtered with an Extensible Stylesheet Language Transformations File (XSLT). Because of this proceeding, no PoC specific version of the UML microwave model was required. Attributes could flexibly be added and removed from the YANG model by tagging them in the UML and re-running the conversion process. And all shortcomings of the microwave model, which were discovered during the PoC preparation, could directly be corrected in the original UML microwave model.

The simplified UML files for the core model and the microwave model were converted by ONF Eagle UmlYangTools into YANG models. All deficiencies in the conversion tools were reported to the Eagle project to enhance functionality and quality. However, when the tests of the configuration interfaces between SDN controller and mediators began, manual modification of the YANG modules were made to reduce risk of blocking parallel implementations and tests.

Resources

The YANG modules of the 2nd MW PoC are available for download at the ONF GitHub (https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/models/24-reducedCoreModel-MWTN-Prio1).

6.4 Evaluation of Open Source Netconf Servers

A mediator is implemented as a stand-alone process running on a server. It typically consists of two parts:

- A Netconf server handles the Netconf protocol, processes Netconf messages based on the Microwave-specific YANG data models, and provides Netconf features like data stores.
- An adaptation layer maps attributes in Netconf messages to/from the proprietary management protocol of the microwave element.

The preferred way for the 2nd MW PoC was to use a common Netconf server framework for all mediators in order to share experience among all participants thus saving development time.

Two open source Netconf server frameworks were selected as candidates: Netopeer [https://github.com/CESNET/netopeer] and OpenYuma [https://github.com/OpenClovis/OpenYuma]. Both frameworks aim at providing extensive support of Netconf features. They are implemented in C programming language and use call back functions for retrieving or updating Netconf attributes from/to the backend, that is, the proprietary microwave element protocol.

The detailed analysis of the capabilities showed that both frameworks are able to support the features required for the PoC. Both frameworks were found to be equivalent except for one severe limitation: On start-up the internal data store of Netopeer cannot be initialized with the current configuration retrieved from the microwave element.

Based on these findings OpenYuma was proposed as basis for the mediators.

6.5 Basis Mediator

Context

Already at the very beginning of the preparation of the 2nd MW PoC, it became clear that all participating vendors will require an external mediator. Since basic framework as well as northbound interface of these mediators is identical for each of the individual implementations, it has been decided to provide a generic Basic Mediator to all the participating vendors. This measure saved significant development and testing efforts.

Implementation

The Basic Mediator is built on OpenYuma and its integrated netconfd agent. During the PoC, its northbound interface (NBI) has been loaded with the reduced ONF YANG model for microwave containing the priority 1 attributes. In general, the mediator is not strictly associated with a specific model or model version. It can easily be recompiled with new YANG models.

The individual southbound interface (SBI) of the mediator had to be added by each vendor and can base on various protocols and proprietary data models. Individual call-backs have to be defined to translate between the standard ONF microwave model on the northbound and the proprietary models on the southbound.

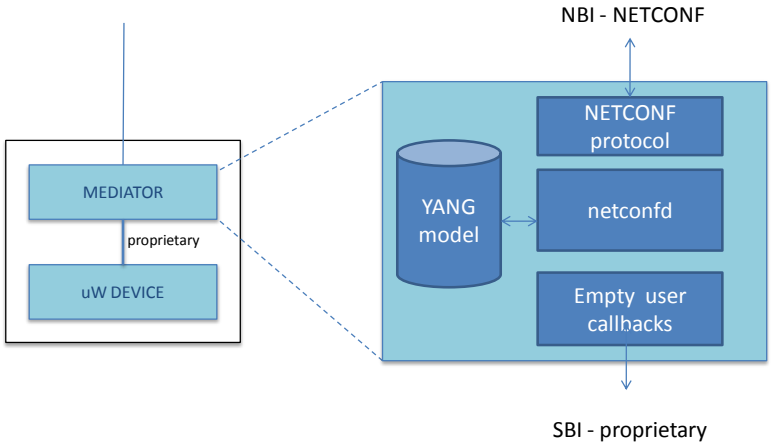

Figure 6: Basic Mediator

Netconf protocol assumes that the client establishes the connection to the server. Hence the Netconf client must know the IP address and login credentials to connect to the mediator. The mediator communicates this information to the SDN controller (Netconf client) by sending a Restconf 'announce' call. This call triggers the SDN controller to access to the Ombudsman (Netconf server). This invitation has been implemented as proprietary code from each vendor.

A YANG module defines the semantics and syntax of a specific management feature. They are similar to SMIv2 (MIB) modules, but much more powerful and extensible. YANG provides the ability to define a detailed programmatic interface utilizing all protocol features:

- reusable derived data types
- reusable groupings of objects
- RPC operations

- database objects
- notifications

The netconfd program is a Netconf-over-SSH server implementation. It is driven directly by YANG files, and provides a robust and secure database interface using standard Netconf protocol operations. All aspects of Netconf protocol operation handling can be done automatically by the netconfd server.

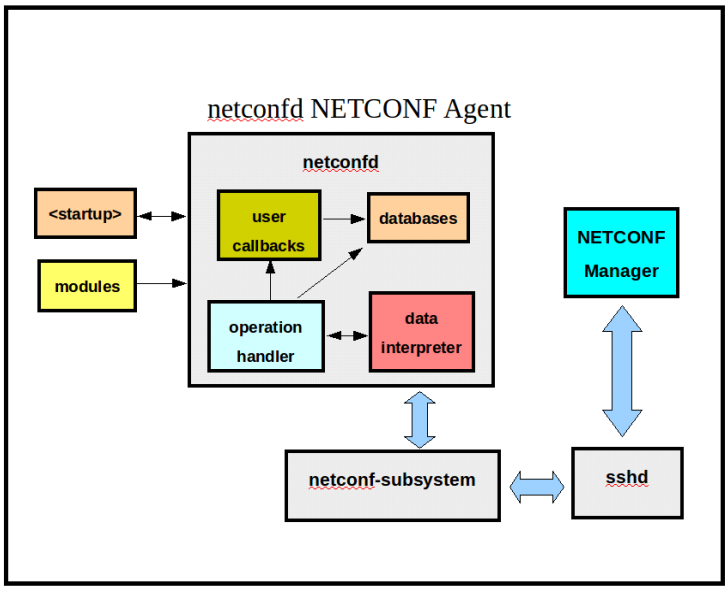

Figure 7: netconfd Server (from open Yuma user manual)

The following notification event types are built into the netconfd server:

- replayComplete Notification replay has ended
- notificationComplete Notification delivery has ended
- sysStartup Server startup event
- sysSessionStart Netconf session started
- sysSessionEnd Netconf session ended
- sysConfigChange <running>configuration has changed
- sysCapabilityChange Server capability added or deleted
- sysConfirmedCommit Confirmed-commit procedure event

The basic mediator framework can be reused in the future to extend the microwave model so far implemented. There are many ways to accomplish this: it is possible to add new YANG modules, it is also possible to update current modules and finally it is also possible to add modules with concurrent version of the model as netconfd support it.

During the development time of the basic mediator, a minor limitation of OpenYuma has been discovered. It is caused by a lack of user callbacks to get the editable parameters. OpenYuma framework generates only the callbacks called to verify and commit the editable parameters. When a manager reads an editable parameter (using Netconf get or get-config rpc) netconfd answers with the value it had cached. From the standpoint of mediator, it is better to read the parameter directly from the device instead of the internal cache, since the value of a given parameter in the device could change after setting another parameter.

OpenYuma provides a suite of tools for development and usage of network management information.

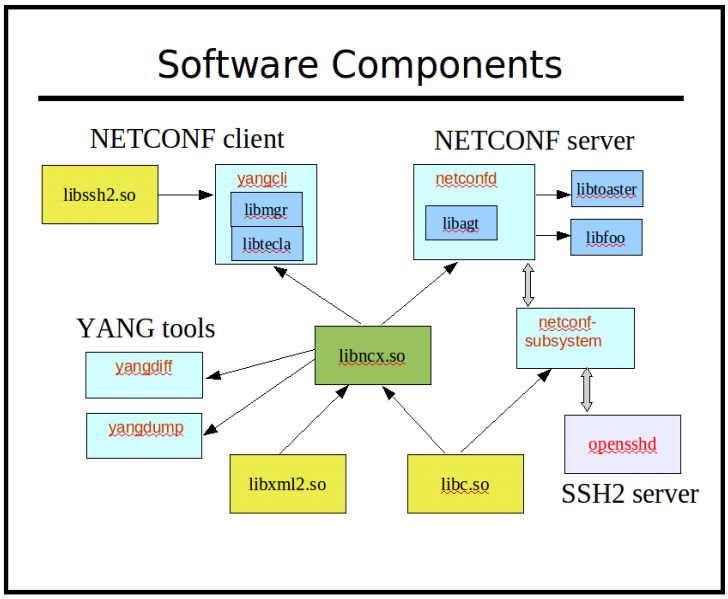

Figure 8: Tools of OpenYuma (from open Yuma user manual)

A useful tool to test the developed mediator is the CLI (yangcli) that behaves as a Netconf manager.

Resources

Instructions for OpenYuma can be found at
https://github.com/OpenClovis/OpenYuma/tree/master/netconf/doc.

6.6 Default Value Mediator

Context

The Default Value Mediator (DVM) is also a derivate of the Basic Mediator. After loading a configuration from a file, it behaves similar to a conventional mediator, which is connected to a real device.

This offers application providers, who do not have access to wireless transport equipment, the possibility of developing and testing their applications without the need of connecting to any actual microwave device.

Implementation

The Default Value Mediator is a Netconf server coded in C, based on the OpenYuma framework, implementing the YANG models used in the 2nd MW PoC. The server provides default values for the YANG models comprised in the following packages: CoreModel-CoreNetworkModule-ObjectClasses, MicrowaveModel-Notifications and MicrowaveModel-ObjectClasses-MwConnection. The values are available for getting and setting immediately after the server starts. Each of the YANG models is implemented as a C shared object and can be dynamically loaded into the server, even at runtime.

The initial configuration for a small number of attributes of the Default Value Mediator is loaded from a file. Modification of those attributes, that are the most important and need to be unique per device, such as the NE Name or the Radio Signal ID, allow deploying of multiple DVMs. Since recompiling the code is not necessary in the case of modifying these parameters (only mediator restart), it is possible to simulate entire microwave networks with minimum effort.

The Default Value Mediator is also able to periodically generate a Netconf dummy notification. The frequency of the notifications is configurable through the aforementioned file.

The server is implemented in a clear, modular way, providing a callback function for each of the attributes of the YANG model that are not present in the configuration file, making the default values easily modifiable, if needed. These modifications, though, require recompiling the specific YANG module and reloading it in the Netconf server.

Resources

Additional information about the DVM, about how to install and use it, and the source files are available at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/mediator

6.7 Element Auto Detection

Context

Element detection indicates the automated discovery of the mediator and associated microwave device by the SDN controller.

Automated detection facilitates use case as for example automated commissioning of new network elements and is also required in case a new controller instance is put into operation and has to learn from the already existing network.

Automated detection is no matter of course, since the device, respectively the mediator, does just contain a Nefconf server, but no client. As a consequence the mediator is only reacting on being addressed by the controller.

Implementation

There are multiple options to overcome the problem of the passive Netconf server in the mediator

- A second interface, which is based on OpenFlow, between controller and mediator
- OpenDaylight initiated discovery through static configuration

- REST based discovery

The first option is not chosen due to the practical limitation in supporting two stacks in the mediator. The second option did not come into question, because maintaining the mediators' and devices' configurations in the controller contradicts the use cases described above.

The REST based discovery has been chosen for the 2nd MW PoC, because of the simplicity of process and implementation. cURL libraries have been applied. These libraries are integrated in the mediators, which are based on OpenYuma and Netopeer, Similar implementations are also possible in mediators based on other frameworks like ConfD. All mediators applied in the 2nd MW PoC incorporate this solution, which has been developed by Wipro.

The following sequence is executed for element auto detection:

- A REST call gets initiated with the help of the cURL library, when the mediator platform gets initialized. The initiation routine within the NetConf framework (OpenYuma or Netopeer or ConfD) triggers that.
- A REST connection request is constructed based on a static mediator configuration file, which contains the IP address and the login credentials of the SDN controller.
- This connection request is received from a Restconf API of the Discovery Module in OpenDaylight. The Discovery Module accepts the REST connection request of the mediator.
- The connection request has the mediator's Netconf connection credentials as payload. Corresponding data store elements inside OpenDaylight are updated. A discovery notification is sent to the Netconf Connection Handler Module inside OpenDaylight.
- After this, a new Netconf Connector is created and mounted as a device by the Netconf Connector Utility in OpenDaylight.
- Along with this, a notification is sent to the Event Manager to subscribe for notifications. The subscription for events is initiated once relevant YANG modules (especially notifications.yang) are loaded in the cache.

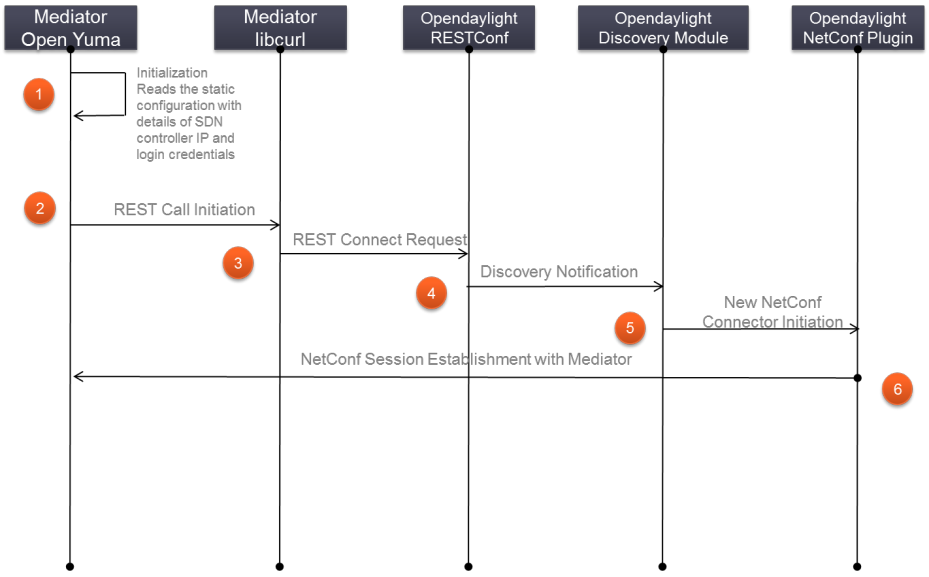

Figure 9: Mediator Discovery Sequence of Operations

Resources

An explanatory readme-file, which contains the detailed steps to implement the discovery mechanism, has been published along with the OpenDaylight code. It can be found at the WTG GitHub.

6.8 UI for Showing and Configuring Parameters

Context

There is a requirement of a user interface (UI) that shows all the network elements, which are currently connected with the controller. This UI shall show configuration, status and capabilities of the network elements according to the ONF microwave information model independently from their type of manufacturer. The UI used in the 2nd MW PoC is also capable of modifying the priority 1 attributes of the AirInterface class.

Implementation

The UI application is built on top of the DLUX GUI capability deployed in OpenDaylight. AngularJS and HTML are the technology stacks used to build the application.

Below are the necessary steps for the application creation:

1. Using the DLUX archetype for creating and building a sample application
2. Deploying this sample application in the OpenDaylight karaf distribution
3. While running the DLUX UI, the sample application should be reflected in the karaf distribution
4. Modifying the sample application with name MWT Config

The application contains the Tab view as mentioned below:

- NetworkElement: Shows all the connected Network Elements
- TerminationPoint: Shows configuration of all the termination points associated with the Network Element
- LayerProtocol: Shows configuration of all the layer protocol associated with termination point
- MW_AirInterface_Pac: Shows configuration of Air Interface associated with a Network Element.
- MW_Structure_Pac: Shows configuration of Structure associated with a Network Element
- MW_Container_Pac: Shows configuration of Container associated with a Network Element
- MW_AI_Config: Provides the GUI to modify the air interface configuration

Proceeding to configure the devices:

1. Click on the MW_AI_Config tab
2. Enter the device name and appropriate layer protocol name
3. Disable the check box ‘Disable Edit Config‘ to edit the configuration
4. Click on the Get button
5. GUI is populated with the current configuration of the device
6. Modify the configuration as per need
7. Click on Set button
8. If Set request is successful, the updated configuration should be reflected when a fresh Get request is made

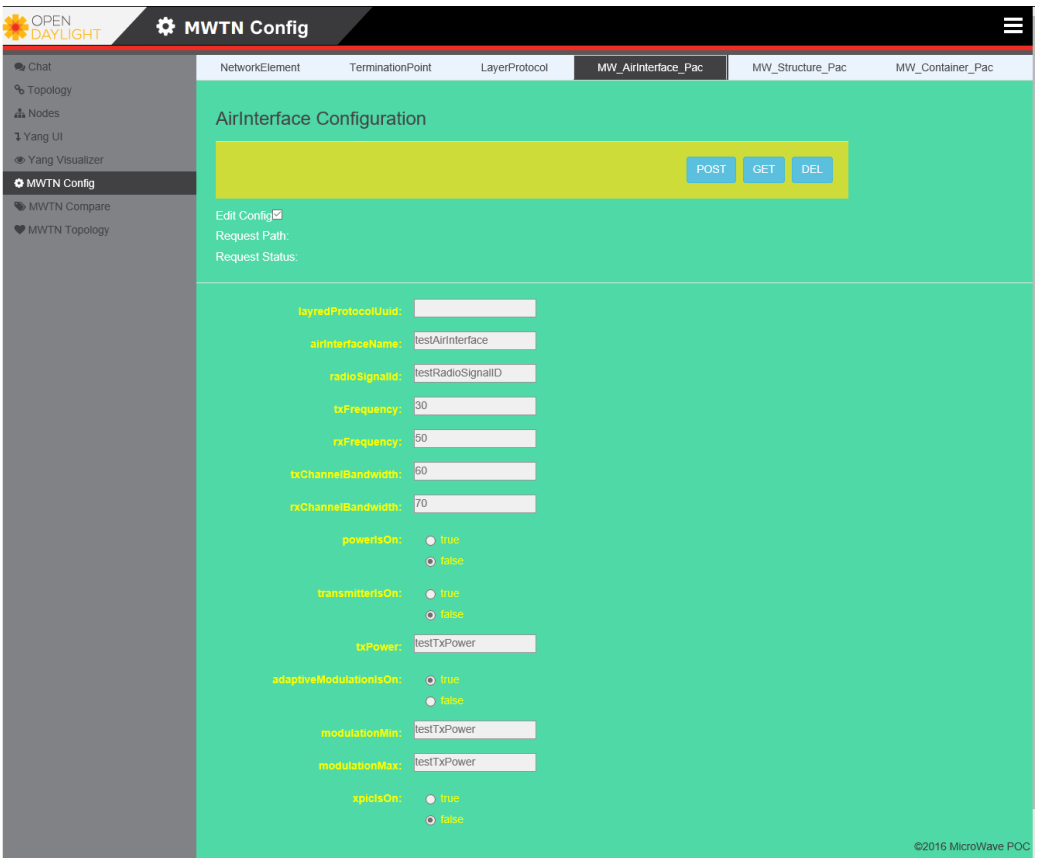

Figure 10: Configuration GUI

Note: In the implementation for the 2nd MW PoC, there is an one to one mapping between layer protocol and termination point.

Resources

The source code of the microwave configuration application is available at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/ux/mwsdnconfigApp

The ready to deploy JAR file of microwave configuration application is available at *https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/ux/deploy*

6.9 Application for Showing Aberrances

Context

Each microwave link is carefully engineered to fulfil the capacity requirements on one hand and to efficiently household with frequency resources on the other. The operator has to apply for the required amount of spectrum at its national regulator. The regulator assigns a specific part of spectrum, which is exactly defined in size and location. Using spectrum different from the assigned might lead to interferences with other links, which result in reduced transport capacity and poor availability of all affected links.

Searching the root cause of interferences is very complex. It's much less complex (but more work) to step through all links and check their configuration. The idea is to automate this process and to run it as a regular routine.

An application with access to the planning databases and the network shall compare the assigned spectrum with the actual configuration of the network elements.

During the PoC, the reference values are read from a simple file (instead from a real planning data base) as a simplification.

Implementation

The application is implemented as OpenDaylight-DLUX web application using the Restconf northbound interface of the controller.

An extract of planning databases was converted into a JSON (JavaScript Object Notation) file. The data structure of the file is already following the structure defined by the YANG models.

Describing an algorithm, which liaises corresponding elements in the file and the network, was one challenge during application development. The current implementation requires that NE-Identifiers (NE-Names) as well as Link-Identifiers (radioSignalIds) are identical in the file and the network. The internal identifiers e.g. for ports or termination points do not have to necessarily be identical to compose the corresponding links from all their objects (e.g. AirInterface, Structure, Container). The model offers all necessary attributes and associations to associate the matching components from both data structures, the file and the network.

The figure below shows the "path" from planning objects of the "MW_Structure_Pac" via the ONF Core Model to the corresponding objects in the network elements.

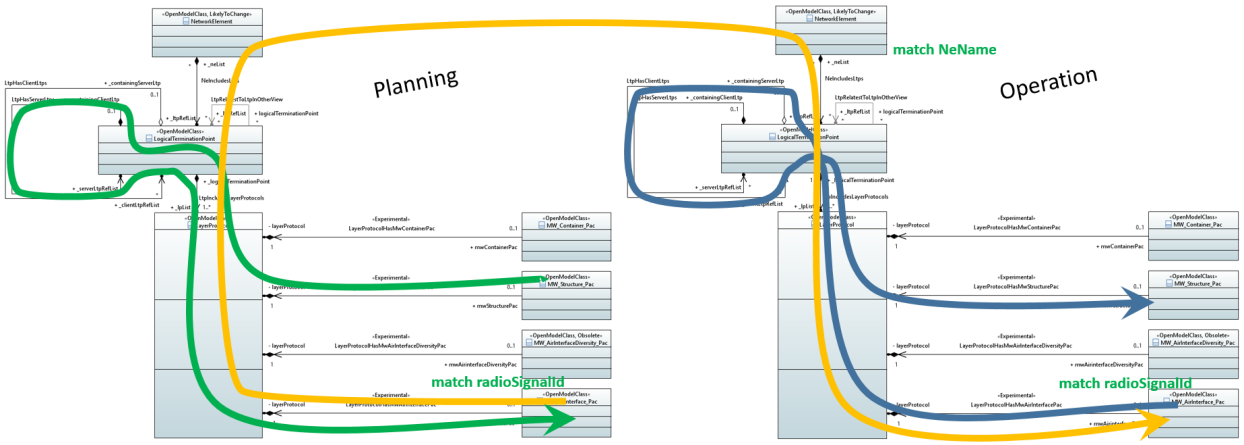

Figure 11: Path from Planning to Actual Object

A table shows the required and actual values in neighbouring columns and highlights the aberrances. In addition, an explanatory text can be display for each attribute, if needed.

In cases the corresponding network element cannot be found, the columns for the actual values are left blank. The application reacts on "connection loss" everts, emitted via WebSockets from OpenDaylight. When the connection between controller and device is lost, all entries

representing actual values disappear. As soon as the connection is established again, these entries are automatically displayed without any manual activity.

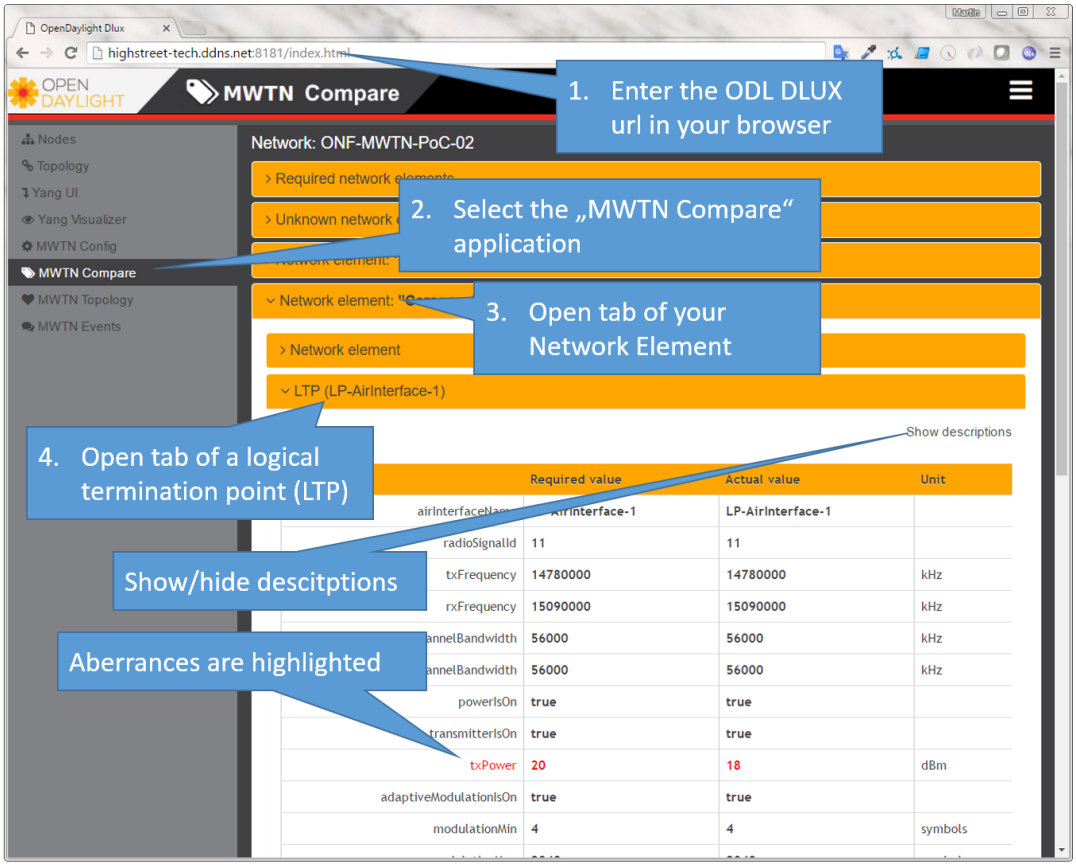

Figure 12: Aberrances User Interface

Resources

The application for showing aberrances to the planning database can be downloaded at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/ux/mwtnCompare.

6.10 GUI for Showing the Configured Network

Context

Already today, the capacity momentarily provided by a microwave link can differ from the planned capacity. This is mainly due to adaptive modulation, which reduces the capacity during times of bad propagation conditions to keep at least parts of the link's capacity alive. This reduction is not intended, but triggered by an extrinsic influence.

SDN will allow to induce additional dynamic into the transport network. The configuration of the microwave devices will become subject to automated, willingly made chances. Switching off one polarization of a dual-polarized link during times of very low traffic demand, could be one example of a willingly made reduction of the capacity.

The configured capacity is not allowed to exceed the planned capacity (which represents also the amount of spectrum assigned by the regulator), and the currently effective capacity can never exceed the configured one.

The configured network is resulting from manual and automated configuring. Information about the effective network also comprises the extrinsic influences.

Wireless Transport Project team decided to implement these two use cases in the 2nd MW PoC, also because it combines reading out the configuration and status attributes of the microwave model with a graphical network view.

Implementation

The application is implemented as OpenDaylight-DLUX web application using the Restonf northbound interface of the SDN controller. The key frameworks are: maven, angular.js, bootstrap and sigma.js.

It shows two different layers of the microwave network:

- (MWPS) AirInterface: physical layer describing the analogue characteristics of the microwave links
- (MWS) Structure: logical structure on the links ready for transporting traffic

MWPS and MWS are independent point-to-point links of different network layers. A separate topology graph is displayed for each layer. In order to create meaningful graphs, MWPS and MWS links are correlated to network elements and sites. The planned network (displayed in grey) is based on an extract of planning databases. In a second step the configured network is displayed in blue as an overlay. In case of the MWS layer also the effective network is displayed. For effective capacities lower than the respective configured capacity, both nodes and links are displayed in red color.

The information contained in the graphs is also displayed in form of tables. One table lists the links the other one shows the nodes of the topology.

The application registers for “connection loss” everts, emitted via WebSockets from OpenDaylight when the connection between the controller and device, respectively mediator is lost or established. If a device gets disconnected, the application assumes that the microwave link is down. Configured and effective connection are removed from the graphs, only the grey connection representing the planning remains. As soon as the connection between device and controller is established again, configured and effective connection are again depicted according to the configuration respectively status information retrieved from the device.

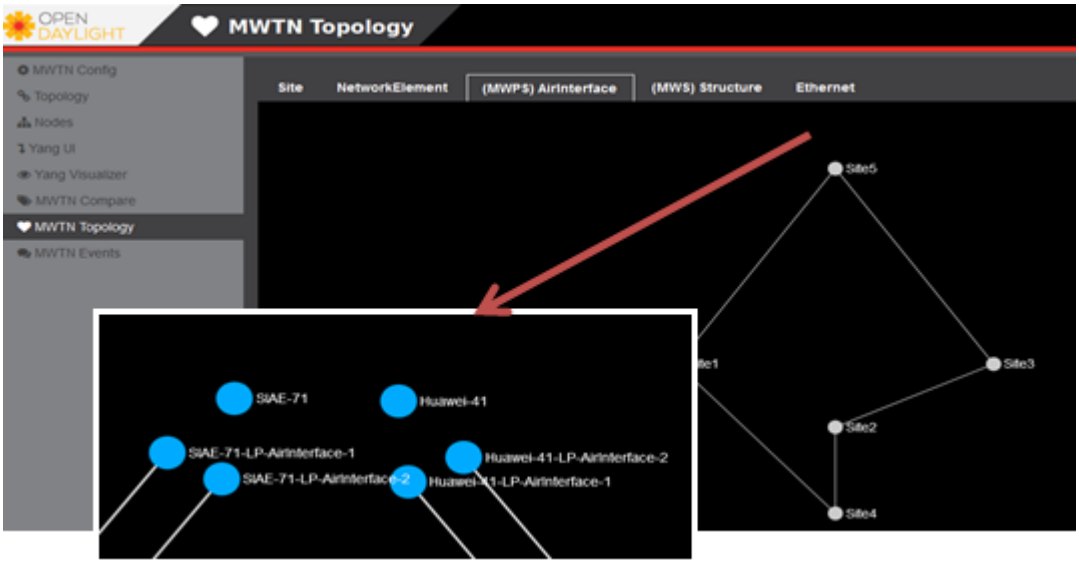

Figure 13: Configured Network GUI

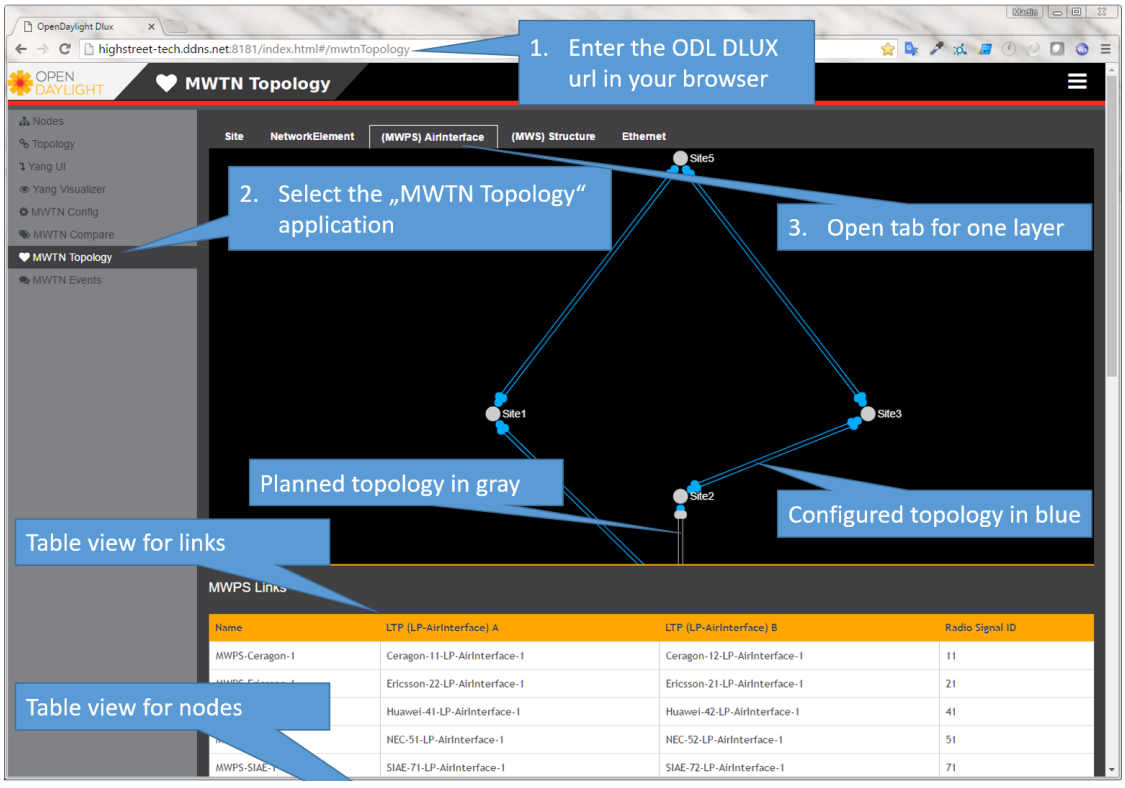

Figure 14: Explanation of Configured Network GUI

Resources

The code of the application for showing the configured and effective network can be found at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/ux/mwtnTopology

6.11 GUI for Showing the Effective Network

Context

Adaptive Modulation and hardware failures are two potential sources of divergences between configured and currently actually effective microwave link capacity. From the operators' point of view, it is important to have an up-to-date picture of the physical capacity in the network to understand the impact on higher network layers.

The Effective Network GUI application gathers all required information in runtime such as the network elements, microwave interfaces and the links' throughput (except operator physical site locations). It displays this information on a graph, which is being updated with live events and statistics. The application is divided into two blocks:

- Microwave Topology Library - provides APIs for applications to retrieve information about microwave attributes of the network (also used by Configured Network GUI application)
- Microwave Effective Network GUI – Answers the 2nd MW PoC's use case "Detection and Visualization of the effective network" by using the Microwave Topology Library and displays all required attributes in a graph

This application demonstrates how any external and controller-independent applications can be easily written and integrated with a live controller by importing the Microwave Topology Library.

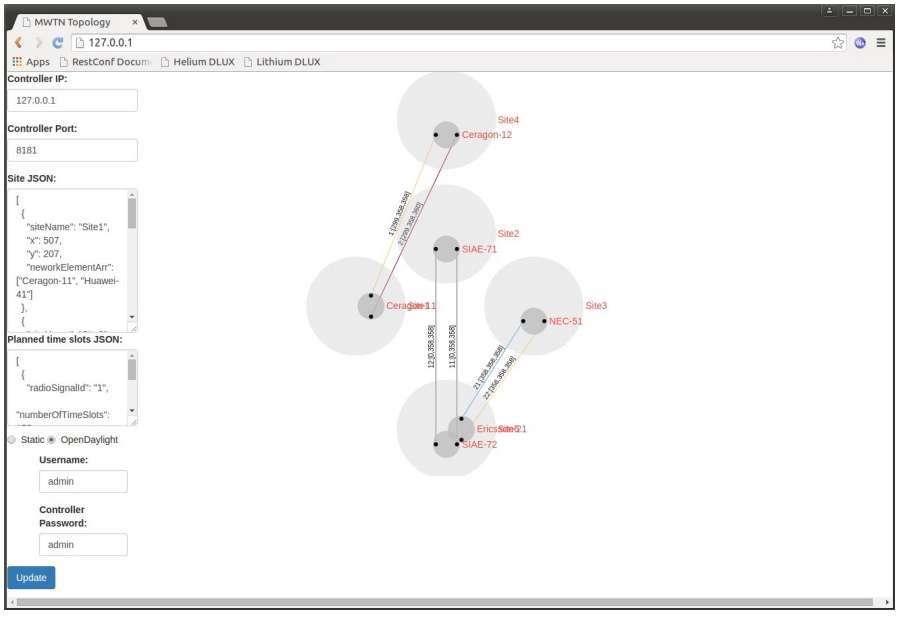

Figure 15: Effective Network GUI

Implementation

The application is a front end application only (no server logic) using Javascript which uses jQuery Ajax to poll the OpenDaylight controller's Restconf northbound interface API.

First, a request to detect all available network elements is sent to the controller by the Microwave Topology Library. Thereafter, the MTL sends a request for the microwave YANG model data of each of the discovered elements. It parses and stores the received information. The GUI application uses the library Javascript APIs to transform this information into Sigma.js graph format and display the network graph in the browser.

Resources

Source code and further details on how to deploy and how to use can be found on https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/ux/mwOperationalTopology

6.12 Event Handling

Context

The experience gained during 1st and 2nd MW PoC confirms that Netconf protocol is much more suitable for configuration tasks than OpenFlow. Netconf defines three types of messages:

- An Invocation is sent from the Netconf client to the Netconf server to request for information or configuration.
- A Reply message is sent from the Netconf server to the client. It contains for example capability or status information of the device. A Reply is always connected to an Invocation by a reference.
- Notifications are sent from the Netconf server without being invoked by the client. They inform about events like for e.g. status changes or alarms.

Current Beryllium release of OpenDaylight comprises a Netconf interface, but it only processes Invocation and Reply messages. Notifications are not supported. Of course, reacting on status changes or alarms is a very important capability in an automated network.

During the 2nd MW PoC the existing Netconf interface has been complemented by Notification.

Implementation

The diagram below shows how events are handled and how they are published to the subscribing application through the northbound interface. All components in green color have been added by Wipro to a forked Beryllium version of OpenDaylight.

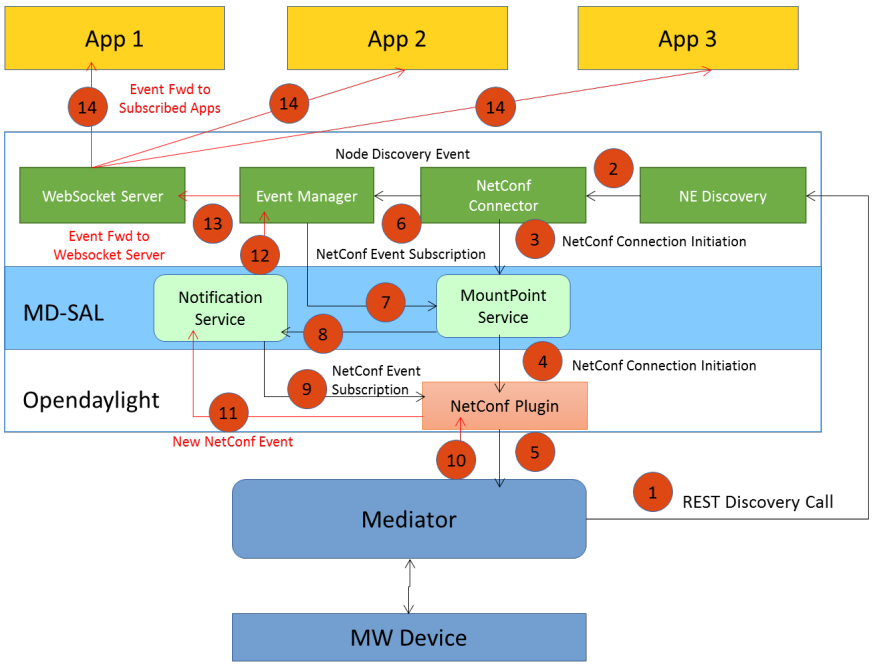

Figure 16: Event Handling in OpenDaylight

OpenDaylight uses standard Netconf event subscription mechanism for notifications. This process is initiated by a Discovery Call sent from the mediator to the controller (step 1). After this, the respective device is mounted in the data store of OpenDaylight.

The Netconf Connector receives the notification message of a new device (step 2) and initiates (step 3) a create_subscription remote procedure call as specified in notifications.yang (step 5) to the mediator.

The respective mediator's Netconf agent registers the OpenDaylight's Netconf plugin (client) as a registered Netconf event subscriber and forwards all the events on a registered channel to OpenDaylight's Netconf plugin. The Event Manager module in OpenDaylight internally subscribes with the Mount Point Service (step 7) available as part of MD-SAL (Model Driven Service Abstraction Layer) in OpenDaylight to forward all the relevant notifications to the mounted device.

After this when the actual event is originated from the mediator, it is forwarded to Netconf Plugin in OpenDaylight by the Netconf agent (step 10) which then is forwarded to Notification Service (step 11) and further to the Event Manager (step 12). The Event Manager decodes the Netconf event and forwards it to the WebSocket Server (step13). WebSocket server publishes the event to all the subscribed clients/applications through a bidirectional WebSocket session (step 14).

Resources

The plugins and event subscription/publishing mechanism in OpenDaylight can be downloaded at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code.

The WebSocket Server for publishing events through web socket channel can be found at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/odl/websocketmanager.

The WebSocket Client Application for demonstrating the web socket based event subscription functionality of Microwave Applications can be found at https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/code/WebsocketClient.

6.13 Testing Framework

Context

A testing framework is aimed at providing integration test capabilities during the development phases of both the models and the applications. It is also effectively leveraged to secure continuous progress during integration of contributions coming from diverse sources.

Testing frameworks are usually operated in multi-site, multi-vendor development projects as they are common within open source communities. For a maximum effectiveness, it should interwork with a configuration management database (CMDB).

The CMDB keeps track of the states of the different contributions and their mutual relationships. This state related information allows reconstruction of former versions of contributions, root cause analysis and change management.

A much desirable feature would be the framework capability of performing both the testing environment setup and the test execution processes in an automated way.

The automatic test environment setup, as well as an automatic execution of the tests in the perspective of a continuous integration of the DUTs shall therefore be properly addressed in the testing framework architecture.

During the test environment setup phase, the continuously released source code artefacts (belonging to YANG models, mediators, SDN controller and applications) are fetched from the CMDB to be built and properly included.

During the test execution phase, a pre-prepared test suite is executed atop the test environment.

The test suite is seen as a modular composition of simple, atomic test cases that can be flexibly expanded during the development and integration process in accord with the evolving use cases evaluation needs.

Implementation

Javascript, as a technological infrastructure for the architecture has been chosen due to its inherent features as well as to the wide choice it provides about runtime environments/libraries targeting a huge set of different purposes.

Github - an open source CMDB - was previously chosen by the Forum for controlled source code sharing and building.

Grunt Task Runner has been selected and leveraged in order to automate tasks, such as setting up and spawning the different testing targets with a single command or building/running the test suite.

Node.js, the well-known runtime environment largely based on JavaScript has been leveraged for the development of the test suite components. Alongside Node.js, other javascript frameworks or libraries have been used: Mocha, Chai, SuperTest, in order to provide Node.js with the testing stack and utilities.

The test suite has been subsequently conceived targeting the following devices under test (DUT) objects:

- The OpenDaylight controller including the added artefacts for configuring, aberrance detection and event handling.
- The YANG data model of the standardized southbound interface of the controller.
- The mediators from the different vendors as well as the Default Value Mediator, which is simulating an attached hardware.

Docker containers are leveraged as virtualized infrastructures for running the mediators and implement the simulated Network Elements topology. The choice of a container instead of a VM is due to its simplicity and ability to grant, in comparison, fair flexibility and performance while keeping a much smaller footprint: such features are best exploited when building large and complex network topologies.

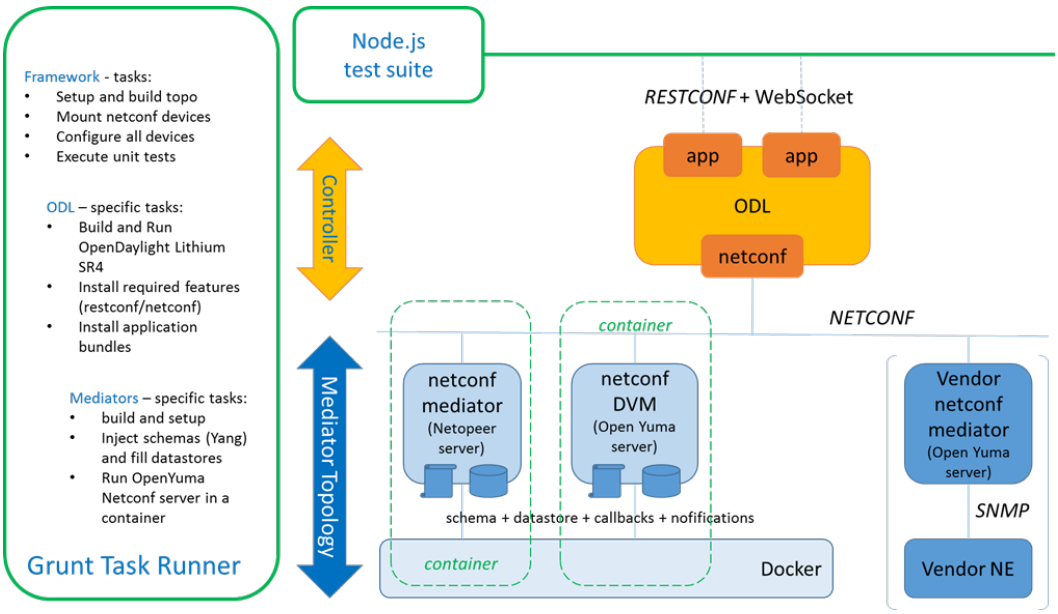

Figure 17: Testing Framework Architecture

Resources

The inherent scalability and the reusability degree of the developed testing framework will be maximally leveraged in the next PoCs in case of developments of the present YANG model. Addenda to the framework would flexibly be supported, as for instance in case of need for implementation and verification of traffic handling oriented use cases.

Framework extension capabilities cover both different and broader use cases setup and broader and more articulated test suites.

The testing framework can be found at:

https://github.com/OpenNetworkingFoundation/CENTENNIAL/tree/master/02-MWTN-PoC/test

System requirements for framework installation on the Linux distribution used within the PoC development can be found in the same folder in *INSTALL.md*. For other general info see *README.md*.

7 References

- Netconf [RFC 6241]
- YANG [RFC 6020]

